ANALYSE BIOMÉTRIQUE DES AILES D'ABEILLES

Rapport de notre projet d'ingénierie 2021

Barbier Arthur, Bauchat Alice, Poulic Chloé, Segal Léontine

Sous le tutorat de Laurent Gerbaud

En relation avec l'association apicole d'Annecy

INTRODUCTION

Notre projet consiste à classifier les abeilles par leur espèce, et particulièrement à distinguer les abeilles de l'espèce Apis mellifera mellifera, plus communément appelées abeilles noires. Ces abeilles sont mellifères et robustes, et représentent donc un avantage pour le monde apicole. Afin de les identifier, nos recherches nous ont mené à 5 caractères morphologiques permettant de différencier les races d'abeilles : la couleur, l'index cubital, la pilosité du 5e tergite abdominal, la largeur du tomentum sur le 4e tergite, et la longueur de la langue. Dans notre projet, nous nous focalisons sur la biométrie des ailes d'abeilles pour réaliser cette classification et ainsi caractériser les essaims dont nous avons les échantillons.

CAHIER DES CHARGES DU PROJET

Pour réaliser ce projet, nous avons d'abord établi un cahier des charges et une stratégie pour répondre au mieux aux besoins des apiculteurs :

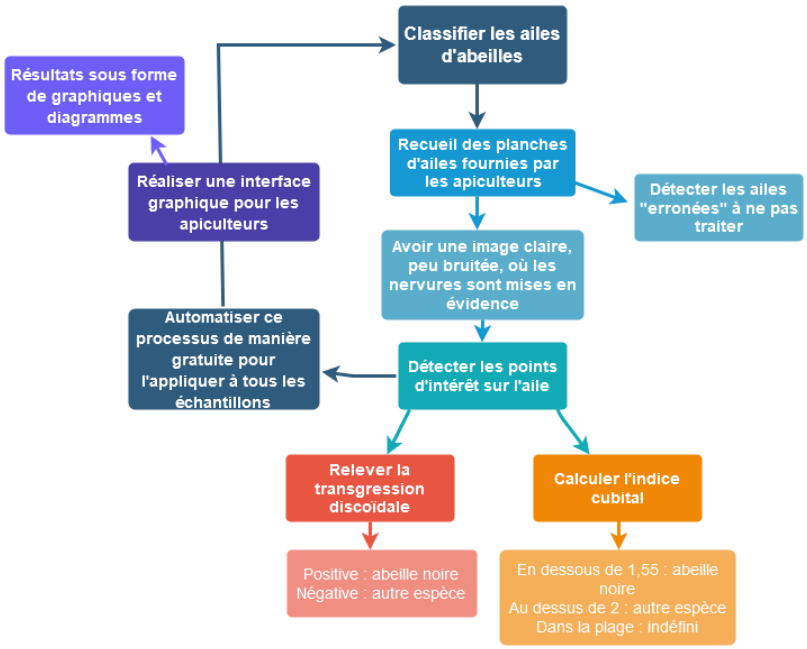


Figure n°0 : Diagramme d'exigences

ACQUISITION DES DONNÉES

Les apiculteurs d'Annecy nous fournissent des images de plaques d'ailes pour l'étude des colonies (sur 15 à 20 individus). L'étape manuelle précède notre projet et est donc importante. Après des échanges sur leurs méthodes et leurs besoins, nous leur avons demandé de présenter les ailes sur une grille de

positionnement (visible sur la figure 1) contenant systématiquement le même nombre d'ailes, scannée par la même machine et dans les mêmes conditions si possible. Tout ceci afin de faciliter l'isolement des ailes que nous traiterons séparément. La pose étant très minutieuse, nous devons prendre en compte dans notre méthode les possibles aléas tels que les poussières, les coupures et pliures sur les ailes, les imperfections des tracés, etc.

Figure n°1 : Positionnement et acquisition des ailes

NOTRE STRATEGIE

Afin de réaliser notre système de classification de manière automatisée et gratuite, nous utilisons Python, un langage de programmation riche en librairies et bien maîtrisé des membres du projet.

Notre objectif est de traiter dans un premier temps les images afin de mieux les exploiter. Il s'agit principalement de supprimer les bruits parasites autour et sur les ailes, de détecter les contours des ailes, les supprimer mais conserver les nervures, sièges des mesures. À partir de cette image modifiée et de l'aile extraite, nous pouvons en 2e temps procéder à la classification en détectant les points caractéristiques, représentés sur la figure n°2, puis de mesurer l'indice cubital et la transgression discoïdale. En recensant ces indices pour l'ensemble des ailes d'une colonie, nous pouvons fournir les graphes demandés par les apiculteurs, comme sur l'extrait de leur fichier Excel fourni sur la figure n°3.

Figure n°2 : points fondamentaux à repérer lors du traitement

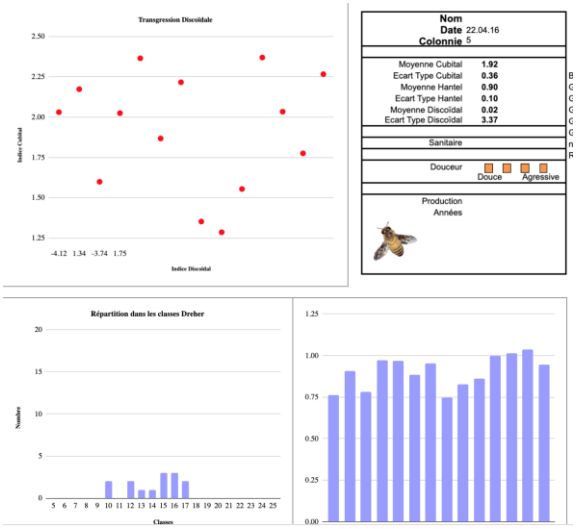


Figure n°3 : extrait des excels finaux des apiculteurs

Nous avons mis en place la stratégie suivante :

Etape 1 : Pré-traitement de l'image

- Filtrer l'image : éviter les points de colle, ne s'intéresser qu'aux nervures
- Ne pas étudier les ailes qui se superposent, celles qui se replient

- Détecter une aile et l'isoler
- L'orienter correctement
- Binarisation (méthode d'Otsu locale)
- Filtrage morphologique : refermer les trous, enlever les formes parasites

Etape 2 : Traitement pour la classification

- Créer une base de données de "token" contenant les 3 points d'intérêt servant à calculer l'indice cubital
- Recherche du pattern contenant ces 3 points dans l'image à analyser
- Isolement de cette partie de l'image
- Binarisation et affinage des traits
- Détection des bifurcations, localisation des points d'intérêt
- Calcul des distances pour déterminer l'indice cubital
- Comparaison des indices cubitaux, élaboration des statistiques au sein de la colonie

PLANNING

La réalisation du projet a été scindée en trois temps :

- La première partie correspond à la **mise en place des algorithmes** qui permettent de pré-traiter les images pour les rendre exploitables et de calculer les paramètres recherchés.

- La seconde partie est la **phase de test**. Nous avons débuté cette phase lorsque nos algorithmes étaient assez avancés, en parallèle de l'avancement sur la suite des méthodes. Les tests permettent de définir les limites et les dysfonctionnements de nos programmes dans le but de les corriger et de rendre leur utilisation plus robuste.

- Enfin, à partir de mars, quand le projet avait bien avancé, nous avons commencé à réfléchir à l'**interface graphique**. En effet, cela permettra de fournir aux apiculteurs des algorithmes simples à utiliser, et qui ne nécessitent aucune connaissance informatique pour être à la portée de tous. Si une approche traitement d'image a ici été choisie comme premier axe de développement, nous sommes conscients que ce projet peut aussi être résolu par une approche nécessitant l'élaboration d'un réseau de neurones. Cet aspect sera ainsi développé dans la troisième et dernière phase, si cela est possible. *(Phase débutée, quasiment terminée pour l'interface graphique, en attente des résultats de l'algorithme)*

PROCEDURE

1. Segmentation
2. Filtrage
3. Extraction des motifs d'intérêt
4. Interface graphique

I. Segmentation

A. Méthode d'identification de rectangles

Imaginons que l'image donnée par l'utilisateur à analyser est celle de la figure n°1.1.

L'objectif est de séparer tous les rectangles en images disjointes pour ensuite en faciliter le traitement. C'est-à-dire que nous n'allons plus traiter une image de grille entière de 14 ailes, mais 14 images d'aile.

L'automatisation de cette étape est primordiale puisqu'elle évite à l'apiculteur un travail pénible de recadrage de 14 images, sur des séries de plusieurs mesures. Cette automatisation se doit d'être suffisamment fiable, puisque pour réaliser des campagnes de relevés, il n'est pas possible de perdre des images.

L'algorithme réalisé repose d'abord sur une détection de rectangles. Ainsi, nous parcourons l'image à la recherche de rectangles. Pour cela nous avons modifié un code déjà existant, trouvé sur le site

https://www.pyimagesearch.com/ par Adrian Rosebrock. L'idée générale est :

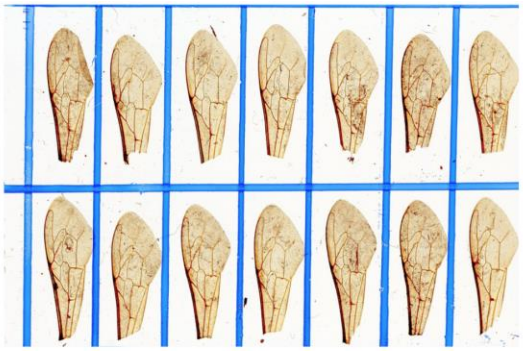


Figure n°1.1 : scan d'ailes fourni par les apiculteurs pour une colonie d'abeilles

- Redimensionner l'image pour que le traitement se fasse plus rapidement. Nous la réduisons donc, suffisamment pour l'accélérer, mais pas trop pour ne pas faire disparaître des éléments de reconnaissance. Afin de travailler par la suite sur des images de qualité similaire à ce qui nous a été fourni, nous n'extrairons pas d'image à partir de cette réduction, mais à partir de l'image initiale.
- Convertir l'image couleur en niveaux de gris, puis nous appliquons un filtrage bilatéral pour supprimer le bruit de l'image, et ensuite nous appliquons un filtre de Canny pour détecter les contours.
- Repérer les contours rectangulaires grâce à la fonction cv2.findContours. Nous choisissons un nombre de contours à traiter pour ne garder que les 10 rectangles les plus grands. Nous effectuons une boucle sur ces 10 contours : calcul des contours approximatifs avec un niveau de précision 1.7% des contours, vérification que le contour a bien 4 coins.
- Vérifier que les rectangles trouvés ont une superficie suffisante. Par suffisante, nous entendons que son aire soit comprise dans l'intervalle $[0.8 * aire_{rectangle} ; 1.2 * aire_{rectangle}]$ avec $aire_{rectangle} = \frac{largeur_{image}}{2} * \frac{longueur_{image}}{14}$ ce qui est une approximation d'un rectangle que nous pensons avoir.

Comme nous souhaitons réaliser cette extraction de contours pour les 14 ailes, nous réalisons cette étape 14 fois. Nous stockons les coordonnées de deux des coins (un du bord supérieur, et un du bord inférieur) dans un tableau.

Nous réalisons ensuite un test pour savoir si nous n'avons pas repéré plusieurs fois le même rectangle, si c'est le cas, nous ne conservons qu'un seul des doublons. Puis nous trouvons le centre des rectangles pour déterminer sur quelle ligne et sur quelle colonne le rectangle se trouve (avec le quadrillage défini comme sur la figure n°1.1). Ainsi, à cette étape nous avons un tableau de 2 lignes et 7 colonnes, dont chacune des cases ne comporte rien si le rectangle lié n'a pas été repéré. Sinon, il comporte les coordonnées des points des coins du rectangle.

Afin de trouver les rectangles restants, nous avons implémenté un algorithme qui permet de délimiter les zones souhaitées grâce à un parcours de l'image et des dimensions des rectangles trouvés. Le principe est le suivant. Si le rectangle non trouvé est dans une colonne dans laquelle un autre rectangle a été trouvé, alors on s'aide de la dimension de celui du dessus (ou du dessous) pour faire uniquement une translation vers le bas (ou vers le haut) des coordonnées. Si dans la colonne les deux rectangles sont manquants, alors on regarde à droite puis à gauche afin de trouver des dimensions. De la même manière, on va translater le rectangle sur la gauche (ou sur la droite) pour trouver le nouveau rectangle.

Si nous réalisons l'algorithme sur l'image de la figure n°1.2. Alors nous avons les résultats suivants :

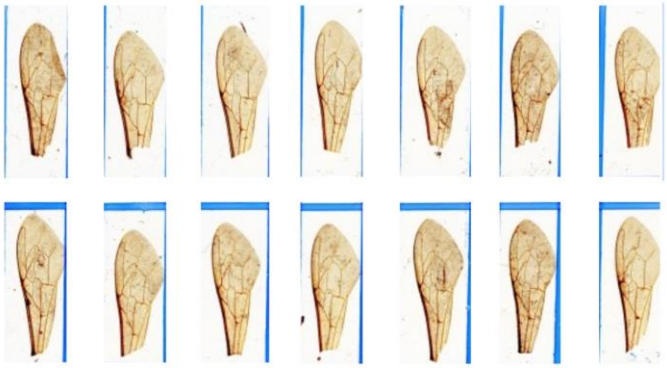


Figure n°1.2 : découpe des ailes

Les résultats sont assez satisfaisants car toutes les ailes sont présentes, il n'y a pas de double, les ailes ne sont pas coupées. Cependant, si nous appliquons l'algorithme sur une autre plaque, les résultats ne sont plus forcément concluants.

En effet, le fait qu'il manque une aile induit un problème de détection et les rectangles que nous allons essayer de trouver vont être de dimensions doubles des rectangles réels. Les traits qui ne sont pas complets sont aussi source de mauvaise détection. Il faut donc une certaine qualité des images pour que l'automatisation fonctionne. De plus, la méthode de recherche des rectangles manquants, après la détection de contours peut poser question. En effet, nous ne tenons pas compte de l'épaisseur des traits, qui est assez importante de toute évidence. Cela peut poser des problèmes de décalages qui vont produire des images erronées si nous décalons de trop au fur et à mesure des extractions comme nous pouvons voir sur la figure n°1.3.

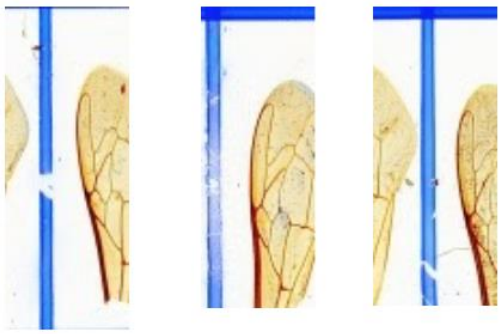


Figure n°1.3 : mauvaise détection

Ainsi, nous avons ajouté à l'algorithme un indice qui définit le décalage. À chaque rectangle, obtenu par identification par rapport aux autres rectangles dans une ligne ou une colonne, sera associé un indice qui indique combien de décalages de « cases » ont été réalisés pour détecter le rectangle qui a permis d'obtenir les coordonnées du rectangle recherché. De cette manière, nous réduisons le décalage final en tenant ensuite compte de cette observation dans le calcul des coordonnées des coins du rectangle.

B. Résultat de cette méthode

Lorsque nous utilisons cette méthode, il y a des résultats très satisfaisants quand le décalage n'est pas important. Cependant, lorsque le décalage devient important, voire maximal (c'est-à-dire 6, il a fallu parcourir toute la ligne pour trouver un rectangle bien détecté), le cadrage n'est pas très satisfaisant comme nous pouvons le voir sur la figure n°1.4. Cependant, la détection reste meilleure que la toute première lorsque nous n'avions pas pris en compte ce décalage, et nous ne trouvons plus d'extraction comme celle de la figure n°1.3.

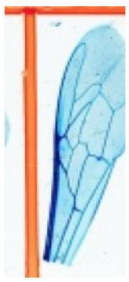


Figure n°1.4 : Exemple d'une aile obtenue avec un indice de décalage égal à 6

Nous pouvons rapporter, en ajout à l'observation précédente, les problèmes suivants. Dans certains cas, l'algorithme ne parvient pas à détecter des formes à 4 coins, par conséquent les rectangles. Nous ne pouvons donc pas traiter ces images. De plus, il arrive aussi que des figures à 4 côtés soient trouvées, mais la superficie n'est pas suffisante pour ressembler à un rectangle qui entoure une aile. Si c'est le cas pour toute l'image, alors il n'est pas possible de traiter cette donnée non plus.

Nous avons implémenté cette méthode sur l'ensemble de la base de données qui nous a été fournie et nous avons obtenu une segmentation convenable pour 68% des ailes. Plus de 20% de l'erreur est dû au décalage vu précédemment, un algorithme plus robuste pourrait augmenter drastiquement le taux de réussite.

C. Autre approche possible : algorithme de Watershed

Initialement, nous souhaitions utiliser une segmentation d'image afin de trouver les ailes. Cette méthode permet de traiter des images où il n'y a pas de grille, ce qui éviterait donc ce tracé, demandant une grande minutie, pour les apiculteurs. Cependant, nous n'étions pas parvenus à utiliser cette méthode.

Récemment, ayant la connaissance de l'algorithme de Watershed, nous pouvons reprendre cette méthode pour détecter les ailes. Le principe est expliqué en annexe mais, globalement, cette technique consiste à faire grossir simultanément toutes les régions jusqu'à ce que l'image soit entièrement segmentée. Cette technique tire son nom d'une analogie avec la géophysique. On peut en effet considérer les valeurs d'intensité des pixels d'une image comme une information d'altitude. Dans ce cas on peut représenter cette image (appelée carte d'élévation) comme un terrain en 3 dimensions. Le principe est alors de remplir progressivement d'eau chaque bassin du terrain. Chaque bassin représente une région. Ainsi cette méthode permet de détecter des objets, auxquels nous pouvons attribuer un label puis trouver l'emplacement de l'objet grâce à des fonctions Python.

Nous avons essayé la première étape qui correspond à la détection des ailes à l'aide de l'algorithme des K-means et celui de Watershed [7]. Le principe est le suivant :

- une première segmentation pour séparer le fond des ailes : nous avons utilisé un filtre Gaussien pour réduire le bruit. Le résultat est suffisamment satisfaisant pour passer à l'étape suivante, bien qu'imparfait. Nous avons binarisé l'image pour simplifier le traitement, puis nous avons fait le choix de deux clusters [8] pour séparer les ailes du fond. Nous obtenons l'image binaire de la figure n°1.5.

Figure n°1.5 : Résultat de la segmentation par la méthode des K-Means d'une image d'aile non quadrillée

-

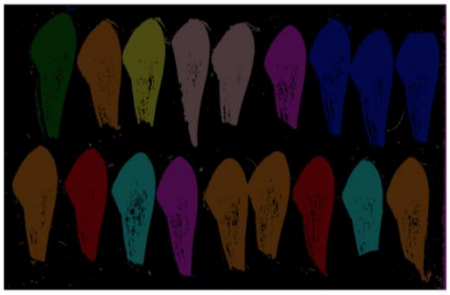


Figure n°1.6 : Extraction des ailes par algorithme de Watershed

Puis nous utilisons l'algorithme de Watershed. Pour faciliter le traitement, nous définissons les marqueurs de fond qui ont une valeur 0, et les marqueurs des ailes qui ont une valeur 1. Puis nous appliquons la fonction watershed [9], associons les zones à des labels, ce qui donne le résultat de la figure n°1.6. Nous pouvons voir que chacune des ailes a été bien détectée. Cette méthode est efficace pour effectuer le traitement recherché : isoler chacune des ailes pour l'extraire et créer une image à partir de chacune des ailes isolées.

L'idée, pour la suite, est de détecter le centre de chacun des objets, et ses dimensions avec les fonctions de skimage.measure [10] puis de découper des rectangles de même centre que le centre des objets, et de dimensions de manière à faire entrer l'objet à l'intérieur. Puis continuer le processus que nous avons réalisé pour la suite du traitement.

II. Filtrage

A. Présentation de l'algorithme

Imaginons que l'image d'aile à analyser est celle de la figure n°2.1.

L'objectif est d'obtenir une image binaire avec seulement les nervures de l'aile car c'est grâce à elles que l'on peut obtenir l'indice cubital. Une bonne filtration est nécessaire car cela permettra une meilleure extraction des données, mais aussi une meilleure précision de mesure. De plus cela permet d'éviter des résultats faux mais aussi de pouvoir traiter toutes les ailes.

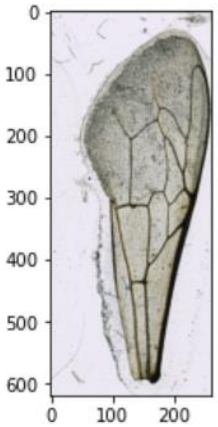
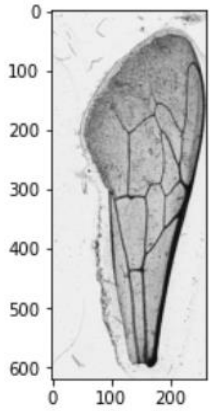


Figure n°2.1 : image à filtrer

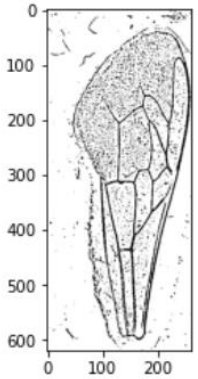
Tout d'abord, l'algorithme va transformer cette image en image à 2D: en noir et blanc. Pour être plus polyvalent en cas de changement de luminosité lors de la prise de l'image, nous allons faire utiliser la transformée générique et ne pas se focaliser sur les couleurs de l'aile.

On obtient alors l'image de la figure n°2.2. On observe que toutes les nervures ont bien été conservées, mais cela fait ressortir un bruit semblable au bruit poivre et sel: de nombreux pixels sont noirs.



Ensuite on va binariser cette image pour pouvoir en extraire les contours. Après avoir testé les différents filtres de binarisation d'openCV, le filtre adaptatif Otsu est le meilleur car il est plus résistant aux changements de luminosité.

Fi de gris



On peut voir que l'on garde quasiment la totalité des nervures, même si on peut voir que le contour le plus à droite qui est utilisé pour la détection de la transgression discoïdale est trop épais et n'est donc pas conservé par ce filtre. Mais on observe que le bruit est encore plus clair maintenant et qu'un filtre passe bas sera inefficace car les points noirs sont à la même échelle que les nervures.

Pour débruiter l'image nous avons d'abord pensé à utiliser les filtres morphologiques. Mais les opérations standard de fermeture avec des masques en forme de cercle ou de traits sont inefficaces. En effet sur l'image de gauche de la figure n°2.4, on observe que la fermeture va enlever une partie du bruit mais aussi très rapidement les nervures.

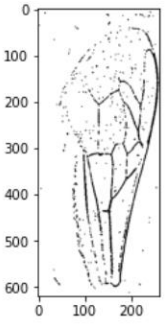


Figure n°2.4 : image filtrée avec un filtre morphologique Closing

L'objectif va alors être de chercher à trouver un pixel de l'aile. Il suffira alors de conserver seulement les pixels noirs adjacents au pixel de l'aile pour obtenir l'image finale. Notre première idée était de faire un algorithme du bassin versant pour délimiter chaque zone fermée de l'image. Mais nous nous sommes rendu compte qu'il suffisait de le faire sur une cellule de l'aile.

L'algorithme va chercher à trouver un point d'une cellule de l'aile. Pour ce faire, le point de départ est défini arbitrairement et est modifié s' il n'est pas dans une cellule (le premier se situe au centre de l'image, s' il ne fonctionne pas on se déplace un peu vers le bas etc.)

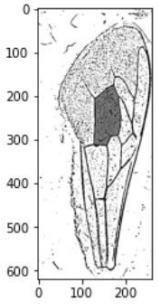


Figure n°2.5 : image en noir et blanc avec une cellule coloriée en gris

Nous allons ensuite marquer tous les pixels blancs adjacents à ce point de la cellule en les coloriant en gris comme

nous pouvons le voir sur la figure n°2.5.

Maintenant à l'aide d'un masque vertical et un masque horizontal, on peut se débarrasser du bruit à l'intérieur de cette cellule comme nous pouvons le voir sur la figure n°2.6. Maintenant en repartant du même point initial on sait que le prochain point noir que l'on rencontre est forcément un point des nervures de l'aile.

Il suffit alors de conserver seulement les points adjacents au point marquant l'aile comme nous pouvons le voir sur la figure n°2.7. On décide de les colorier en gris pour les différencier des autres pixels noirs.

Finalement on ne conserve que ces pixels gris et on obtient l'image filtrée avec seulement les nervures(figure n°2.8).

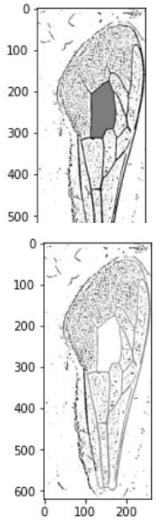


Figure n°2.7 : image en noir et blanc avec la partie intéressante en gris

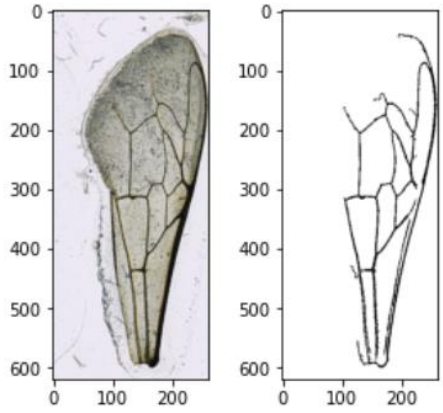


Figure n°2.8 : image filtrée

B. Résultats et statistiques

On peut voir que le filtrage est très satisfaisant de n'obtenir que la partie intéressante. Lorsqu'on lui donne une image sans feutre et bien cadré on obtient une réussite de 100%.

Nous avons implémenté cette méthode sur l'ensemble des images d'ailes bien segmentées et on obtient un taux de réussite de 65%.

En effet la segmentation va donner des images d'ailes légèrement décalées, de plus les traits de feutres peuvent perturber la binarisation de l'image et donc le filtrage.

Cependant après la segmentation, certaines ailes sont décalées,

De plus pour filtrer 14 ailes(un scan) cela prend en moyenne 57sec. Cela peut sembler assez long, mais pour caractériser une ruche il faut de 20 à 100 abeilles soit un temps de calcul maximum de 7 min. Mais ce temps est drastiquement réduit lorsque l'image est bien cadrée car on trouve dès la première itération une cellule de l'aile. Cela réduit le temps de calcul de moitié environ.

C. Autre approche : watershed, reconstruction géodésique

On pourrait essayer de trouver un autre filtre à appliquer à l'image en noir et blanc qui conserve plus d'information (comme celle qui permet de trouver la transgression discoïdale). Regarder les résultats du filtre de Yen de la librairie skimage.

Une autre idée serait d'utiliser l'algorithme du "bassin versant" pour isoler chaque cellule de l'aile. On pourrait alors regarder la surface couverte par chaque zone pour différencier une cellule du fond de l'image.

Enfin utiliser la reconstruction géodésique (fonction reconstruction de skimage) permet d'obtenir le résultat de la dernière étape beaucoup plus rapidement en utilisant l'image après le filtre binaire et l'image "seed" avec pour seul point noir le point de l'aile trouvé.

III. Extraction des motifs d'intérêt

A. Identification et recherche

Contexte Déterminer l'indice cubital et la transgression discoïdale d'une aile donnée nécessite d'obtenir les coordonnées de certains points d'intérêt de cette aile.

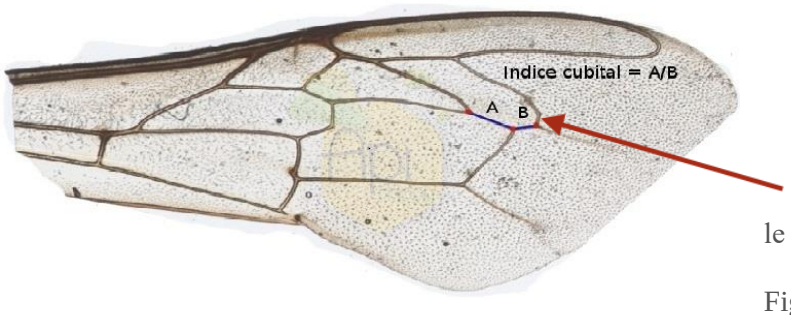


Figure n°3.1 : Points d'intérêt à extraire dans le cas du calcul de l'indice cubital

3 points d'intérêt pour le calcul de l'indice cubital

L'équipe de l'année précédente avait proposé un algorithme construit par rapport à la forme bien spécifique d'une seule et unique aile (orientation des nervures, nombres de nervures après filtrage…). Nous voulions donc écrire un algorithme généralisable à toute aile.Pour cela, nous avons proposé une approche différente pour obtenir les coordonnées de nos points d'intérêt : bien que toutes les ailes soient différentes, le motif d'intérêt pour obtenir l'indice cubital est lui, toujours le même au sein des abeilles noires, comme on le constate ci-dessous (motifs issus de la ruche n°18) :

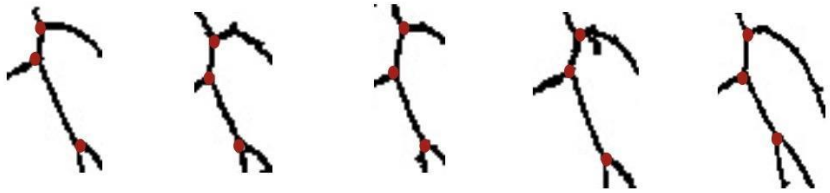


Figure n°3.2 : parties contenant les 3 points d'intérêt sur différentes ailes

La problématique qu'est l'**extraction de points d'intérêt devient donc avant tout une problématique de reconnaissance de pattern**. En effet, une fois le motif extrait, il sera plus évident d'écrire un algorithme généralisable à toute aile, s'il ne traite uniquement que les motifs (et non la totalité de l'aile). **Dans cette partie, nous développerons cette étape d'extraction de pattern, appliquée au cas du motif « indice cubital » seulement** (on procédera de la même façon pour la recherche du motif « transgression discoïdale »).

Dans un premier temps, l'algorithme écrit sera présenté (a) Algorithme et Corrélation de phase).
La démarche concernant l'élaboration des jeux de tests, d'identification et résolution des failles sera ensuite développée (b) Phase de Tests et résultats).
Pour conclure, on propose quelques pistes envisageables pour surmonter quelques difficultés subsistances (d) Améliorations possibles de l'algorithme).

a) Algorithme et Corrélation de phase

Choix de la méthode

Pour mener à bien une reconnaissance de pattern, il est possible de mettre en place plusieurs méthodes. A notre connaissance, l'élaboration de deux méthodes étaient envisageables :

- la première, a recours à la **corrélation de phase**.
- La seconde, consiste à développer un **réseau de neurones**, entraîné à reconnaître notre motif d'intérêt.

La seconde méthode a rapidement été abandonnée car le bon fonctionnement d'un réseau de neurones repose avant tout sur la **qualité de la base de données d'entraînement fournie**. Cette base de données d'entraînement n'existant pas, nous avons d'abord pensé à la créer. Cependant, le nombre de données fournies jusqu'à ce jour (scans de différentes ruches) nous permet de créer une base de données d'entraînement de **quelques centaines de données**, **ce qui serait insuffisant pour assurer la précision de l'extraction de pattern**.
La première méthode a donc été retenue.

La **corrélation de phase**, calculée entre deux images *image 1* et *image 2*, consiste à **estimer leur ressemblance** dans la mesure où les coordonnées de la valeur maximale de la "map" résultante correspond à la translation à effectuer pour retrouver l'*image 1* à partir de l'*image 2*. La corrélation temporelle, correspondant à une simple multiplication fréquentielle, il est plus aisé de travailler dans le domaine spectral

que temporel.
La **corrélation de phase** effectue ainsi le calcul suivant :

$$C(img_1, img_2) = TF^{-1}\left(\frac{TF(img_1)}{|TF(img_1)|} . \left(\frac{TF(img_2)}{|TF(img_2)|}\right)^*\right)$$

La normalisation par les modules des transformées de Fourier permet d'uniquement récupérer l'information concernant la phase (c'est-à-dire, concernant la translation temporelle) et non concernant le module.
La **corrélation de phase**, bien que robuste face au bruit potentiellement présent dans les deux images, est toutefois sensible aux changements d'échelle et aux rotations du pattern.

Algorithme (cf. ***"phaseCorrelation.py"*** fourni dans le dossier **code**)

On développe ici le fonctionnement de l'algorithme à l'aide d'un exemple illustratif

L'algorithme basé sur la **corrélation de phase** repose sur la **création préalable de deux bases de données** :

- Base de tokens de référence : elle comporte un ensemble de motifs "type" (appelés **tokens**), images des motifs de référence que l'on cherche à reconnaître sur une aile quelconque.
-

  Ces tokens ont été créés manuellement à partir des images de ruches fournies par les apiculteurs. Chaque token à été créé de la façon suivante :
  - on extrait, manuellement, une aile de l'image de la ruche originale,
  - On la filtre à l'aide de l'algorithme décrit dans la partie précédente **Filtrage**
  - On découpe de nouveau, manuellement, le motif d'intérêt.

  On réitère cela autant de fois que l'on veut de tokens dans la base de données.

- Base de données d'ailes "inconnues" : elle comporte un ensemble d'ailes filtrées (algorithme présenté dans la partie précédente **Filtrage**) et constitue l'ensemble des ailes dont on cherche à extraire leur motif d'intérêt.

Une fois ces deux bases créées, l'algorithme se déroule de la façon suivante :
- Pour chaque aile de la base Base de données d'ailes "inconnues" : on calcule sa transformée de Fourier

- On parcourt la base Base de tokens de référence : pour chaque token de cette base, on calcule sa transformée de Fourier et on récupère les coordonnées du maximum de la **corrélation de phase** calculée entre la TF de l'aile inconnue et celle du token.

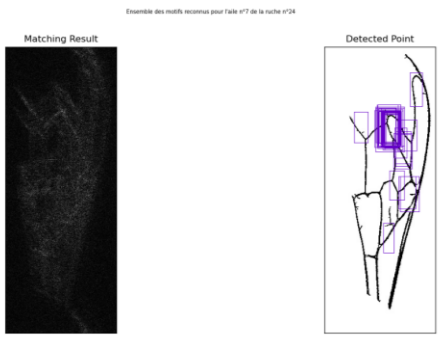


Figure n°3.3 : Etape 1 de l'algorithme appliqué à l'aide n°7 de la ruche n°24

Chaque rectangle violet correspond à la reconnaissance d'un token de la base de données Base de tokens de référence (figure de droite). On voit ici que la majorité d'entre eux ont été reconnus dans la même zone. L'image de gauche correspond au résultat du calcul de la corrélation de phase entre l'aide et le dernier token de la base Base de tokens de référence.

- On crée une nouvelle image, appelée "**montagne**" qui va associer chaque maximum de la fonction de **corrélation de phase** à une **gaussienne 2D**, permettant ainsi de **cartographier la probabilité de présence du token** sur l'image.

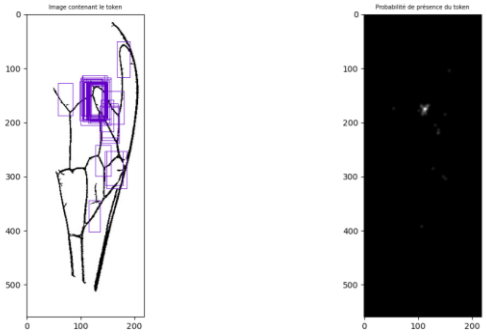


Figure n°3.4 : Etape 2 de l'algorithme appliqué à l'aide n°7 de la ruche n°24

L'image de droite est l'image "montagne" (pour un sigma de 2 pour la fonction gaussienne). On constate que la probabilité de présence du motif recherché est maximale autour des coordonnées [ligne n°170, colonne n°110].

- On récupère les coordonnées du maximum de cette image "**montage**" et on les associe à celles du motif recherché.

Figure n°3.5 : Etape 3 de l'algorithme appliqué à l'aide n°7 de la ruche n°24

- On récupère les dimensions du token le plus "ressemblant", c'est-à-dire dont les

coordonnées du maximum de la corrélation de phase sont les plus proches de celles du maximum de l'image "montagne". On découpe ainsi le motif identifié suivant ces dimensions récupérées. Le token est enregistré automatiquement dans une nouvelle base de donnée "token trouvés"

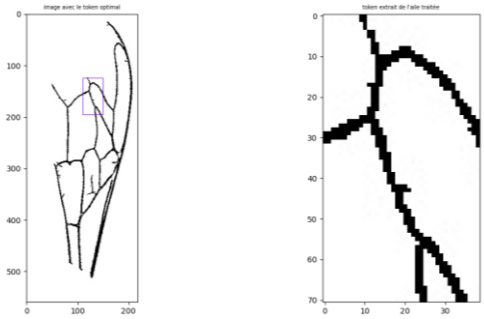


Figure n°3.6 : Etape 4 de l'algorithme appliqué à l'aide n°7 de la ruche n°24

On constate que le motif extrait est bel et bien celui qui était initialement recherché (figure de droite).

b) Phase de Tests et résultats

Au delà de l'implémentation de l'algorithme, c'est davantage la base de données Base de tokens de référence qui joue un rôle décisif dans la détection du motif d'intérêt : il s'agit ici d'estimer et de comprendre l'impact du nombre de ses token et de leur nature sur le résultat de l'extraction de motif.

En effet, bien que tous similaires, les tokens peuvent être de nature différente :

- Ils peuvent être plus ou moins "bruités" :

- D'orientation différente suivant la manière dont à été photographiée l'aile :

- Leur forme peut aussi être aussi diverse...

La démarche de construction de la base Base de tokens de référence est la suivante :
On sélectionne pour commencer qu'un token (4), le plus idéal possible, c'est-à-dire non bruité, d'angle de rotation nul et de forme "neutre".
On lance le programme et on constate le nombre d'extractions correctes.
On ajoute un nouveau token de référence tant que l'extraction n'est pas juste à 100%.

L'enjeu est de **minimiser le nombre de tokens** de la base Base de tokens de référence pour **minimiser le temps de calcul de l'algorithme** (boucles imbriquées de complexité $n.n + n$) tout en s'assurant que les tokens choisis suffisent à représenter au moins leur propre ruche.

Dans le cas de la ruche n°18 (comportant 18 ailes), on a ainsi les résultats suivants :

Pour 1 token dans la base Base de tokens de référence :

15 extractions correctes sur 18

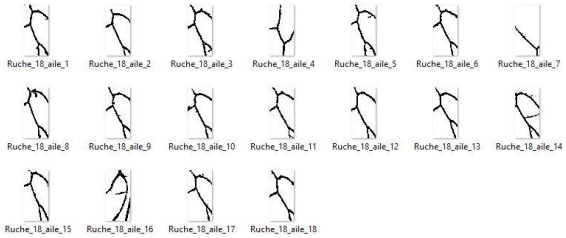


Figure n°3.7 : Motifs extraits de la ruche n°18 à l'aide d'un token de la ruche n°18

Pour 2 token dans la base Base de tokens de référence

17 extractions correctes sur 18

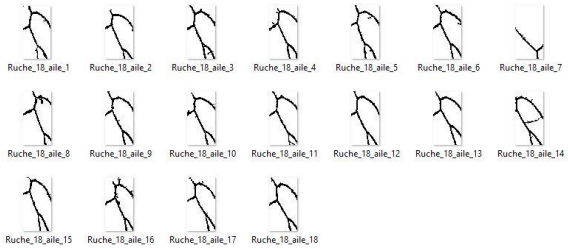


Figure n°3.8 : Motifs extraits de la ruche n°18 à l'aide de deux tokens de la ruche n°18

Pour 3 token dans la base Base de tokens de référence

18 extractions correctes sur 18

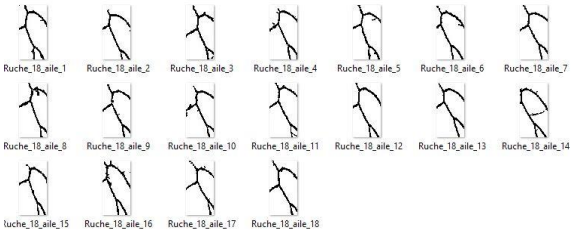


Figure n°3.8 : Motifs extraits de la ruche n°18 à l'aide de trois tokens de la ruche n°18

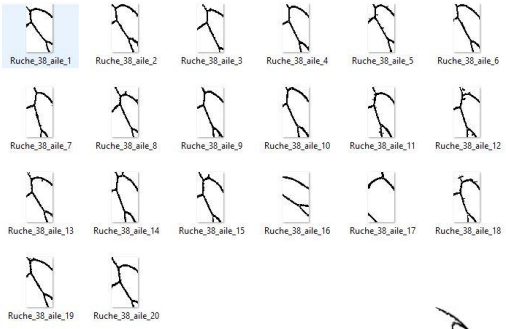
On applique de nouveau l'algorithme à l'aide de cette base, composée de 3 tokens de la ruche n°18 pour extraire les motifs d'intérêt d'une ruche différente, pour vérifier que les tokens idéaux choisis sont aussi représentatifs d'une ruche différente. On étudie la ruche n°24, composée de 20 ailes. Le résultat est très satisfaisant car on obtient 20 extractions justes sur 20. Cependant, si on essai de nouveau avec une ruche dont les ailes ont été placées de façon moins ordonnée (cas de la ruche n°23), les résultats sont mauvais (8 extractions justes sur 12). On augmente de nouveau la taille de la base de données et on réitère ce procédé avec les diverses ruches mises à notre disposition.

On présente ci-dessous les résultats obtenus, permettant d'assurer le meilleur compromis entre **minimisation du nombre de tokens** et **maximisation du nombre d'extractions correctes**. La base de données Base de tokens de référence contient 20 tokens.

N° Ruche	Temps de calcul (sans affichage)	Nombre d'extractions justes
59	9.59 s	20 ailes /20 ailes
21	3.45 s	7 ailes /7 ailes
23	5.38 s	12 ailes /12 ailes
24	8.95 s	20 ailes /20 ailes
18	10.08 s	18 ailes /18 ailes
38	9.05 s	18 ailes /20 ailes

Le cas de la ruche 38 est intéressant car il illustre la sensibilité et la faiblesse de la corrélation de phase à la rotation du motif recherché. En effet voici les motifs extraits de la ruche 38 à l'aide de la base de données Base de tokens de référence :

Les deux mauvaises extractions viennent de l'aile 16 et 17 de la ruche 38.



Observons individuellement le cas de l'aile 17 :

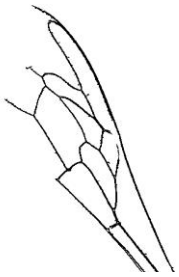


Figure n°3.9 : Aile n°17 de la ruche n°38

Cette aile a subi une rotation particulièrement importante et cette rotation détériore la qualité de la détection et empêche ainsi l'extraction du motif d'intérêt. Le cas de l'aile n°16 est semblable.

c) Améliorations possibles de l'algorithme

L'algorithme actuel est donc fonctionnel dans la mesure où l'extraction du motif d'intérêt est correcte mais plusieur points sont à améliorer :

- **Les temps de calculs sont longs**. Pour 20 ailes à traiter, on approche les 10 secondes pour extraire les motifs d'intérêt. Ces temps sont bien trop long sachant que cette étape du traitement n'est qu'une parmi les trois autres (segmentation, filtrage, calculs de l'indice cubital). L'algorithme pourrait ainsi être optimisé.

- On a été confronté à la **sensibilité de la méthode aux orientations** des ailes. Plusieurs pistes ont été abordées pour résoudre ce problème, tel que le passage en coordonnées polaires : la rotation linéaire devient, en effet, une translation polaire et il est alors possible de réutiliser la corrélation de phase.
  Le code inspiré de celui-ci [11] (cf. ***"PhaseCorrelationRotation.py"*** fourni dans le dossier **code**) est à ce jour en ébauche et non pas assez robuste, mais l'exemple ci-dessous présente le traitement de l'aile problématique n°17 de la ruche n°38. Bien que la détection s'effectue, le token extrait est flou (fonction resize de CV2 utilisée). Cependant, comme l'orientation des ailes seront par la suite imposée et nulle (cf. partie **Segmentation**), il n'est pas fondamental de se préoccuper de cet aspect.

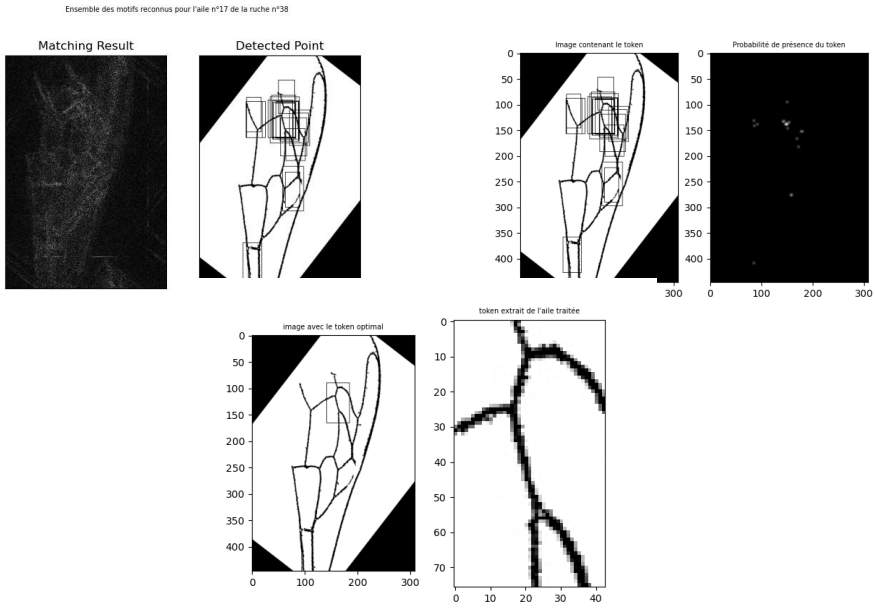
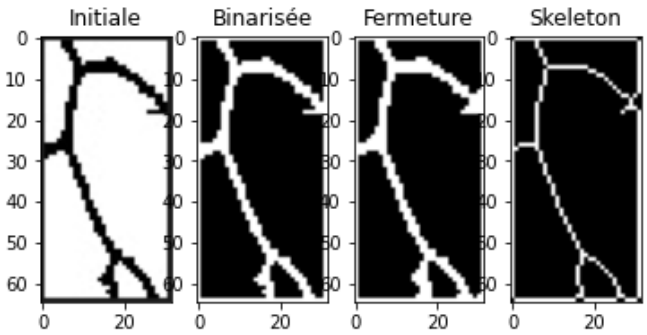


Figure n°3.10 : Résolution du problème de rotation de l'aile n°17 de la ruche n°38

B. Traitement du token, relevé précis des points d'intérêt et calcul de l'indice

Afin de pouvoir déterminer de manière précise les intersections entre les nervures correspondant à nos points d'intérêt, il est nécessaire d'affiner les contours extraits de nos images réduites enregistrées au cours de l'étape précédente. Pour cela nous avons utilisé l'opération morphologique du module skimage.morphology "skeletonize", qui réduit l'épaisseur des contours à 1 pixel sans dégrader les formes, sur une image précédemment binarisée. De plus, en testant des opérations morphologiques telles que la fermeture, nous sommes parvenus à réduire les débuts de nervures parasites qui faussent la détection des intersections. C'est pourquoi nous avons procédé dans l'ordre suivant : *Binarisation -> Fermeture -> Amincissement*



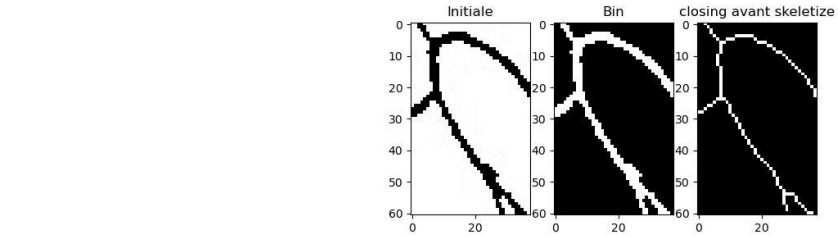


Figure n°25 : Aperçu des opérations morphologiques successives

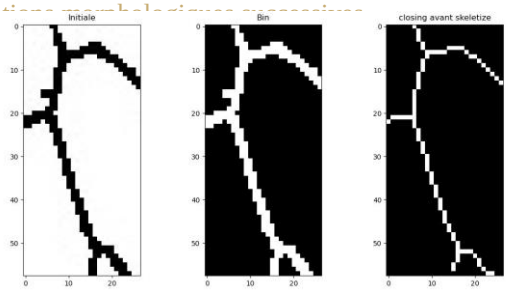


Figure n°26 : Suppression efficace des nervures et épaisseurs parasites

L'image obtenue à la fin nécessite d'être re-binarisée, et nous avons aussi supprimé son cadre pour avoir moins de soucis par la suite :

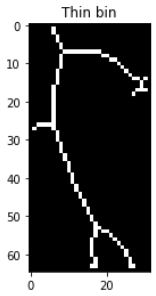


Figure n°27 : Image sur laquelle nous appliquons notre algorithme de détection des points

Pour détecter les 3 points nous servant à calculer l'indice cubital, nous recherchons les intersections grâce au "crossing number", un critère que nous avons calculé pour chaque pixel, qui marche comme suit :

"Si je suis blanc, et que 3 de mes 8 pixels alentours sont blancs, alors je suis un point d'intersection"

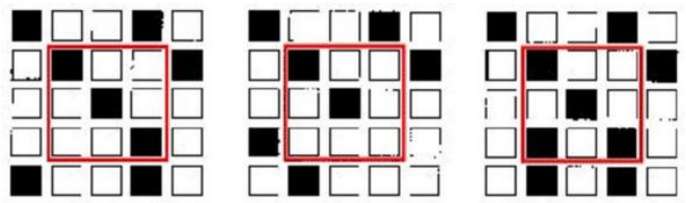


Figure n°28 : Exemple de détection de bifurcation

Ci-dessus, seul le pixel central de la 3ème image est au centre d'une bifurcation : on ajoute donc ses coordonnées à la liste "intersections".

Ceci nous permet de dresser la liste des intersections, soit les points candidats aux 3 points d'intérêt que nous cherchons. Nous remarquons que notre liste contient plusieurs tuples de coordonnées vraiment proches, qui correspondent au même point d'intérêt. Après avoir créé trois groupes dans les meilleurs cas, où chaque groupe de tuples correspond à la position d'un point d'intérêt, nous prenons la moyenne de ces coordonnées pour déterminer les coordonnées finales des trois points d'intérêt.

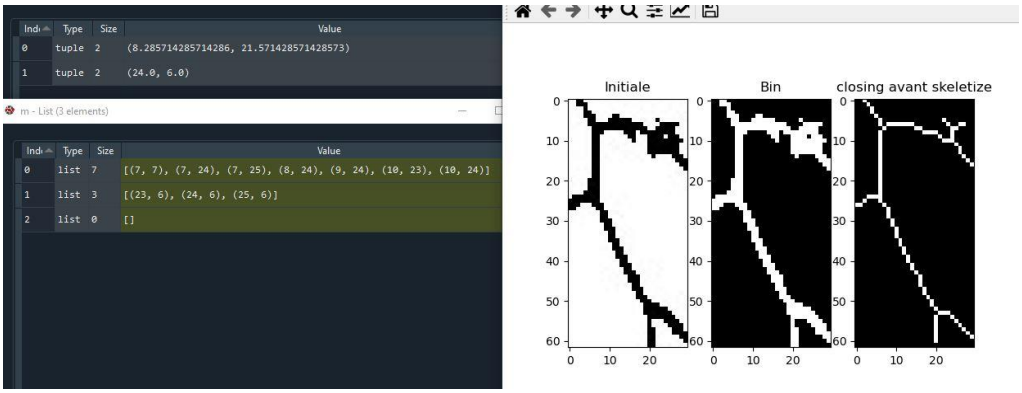


Figure n°29 : Cas de double erreur : un point supplémentaire a été pris en compte dans le premier groupe, et un groupe est vide donc il manque un point

Nous avons alors un tableau avec les coordonnées de 3 points : il ne reste alors plus qu'à calculer la distance euclidienne entre les points les plus proches, puis le rapport des deux distances.

Ruche 18						
Numéro aile	**Indice cubital donné par l'algorithme**	**Indice cubital calculé manuellement**	**Erreur relative**	**Identification de l'abeille noire par rapport à l'indice cubital informatique**	**Identification de l'abeille noire par rapport à l'indice cubital manuel**	**Cohérence de l'identification de l'espèce**
1	33,36352799	1,65	31,71352799			
2	"La détection a échoué"		#VALEUR!			
3	"La détection a échoué"		#VALEUR!			
4	2,08461975	2	0,08461975	négatif	négatif	satisfaisant
5	Intersections non détectées		#VALEUR!			
6	2,137059839	1,8888	0,248259839	négatif	non concluant	neutre
7	1,94161985	2	0,05838015	non concluant	négatif	neutre
8	2,164816785	2,2857	0,120883215	négatif	négatif	satisfaisant
9	1,531305667	2,25	0,718694333	positif	négatif	insatisfaisant
10	1,923814639	2,024	0,100185361	non concluant	négatif	neutre
11	1,903878783	2,25	0,346121217	non concluant	négatif	neutre
12	51,26597312		51,26597312			
13	Intersections non détectées		#VALEUR!			
14	Intersections non détectées		#VALEUR!			
15	Intersections non détectées		#VALEUR!			
16	1,043731166	2	0,956268834	positif	négatif	insatisfaisant
17	"La détection a échoué"		#VALEUR!			
18	1,62459276	1,6	0,02459276	non concluant	non concluant	satisfaisant

Figure n°30 : Tableau comparatif pour l'identification des espèces par l'indice cubital sur la ruche 18 comportant 18 individus

Pourcentage d'ailes non exploitables	Erreur relative moyenne dans les cas opérationnels	Pourcentage d'erreur par rapport à l'identification de l'espèce
50	0,29533394	22

Figure n°31 : Etude du taux d'erreur par notre méthode automatisée

En comparant nos résultats informatiques aux résultats relevés manuellement sur les ailes, on se rend compte que seules 50% des ailes sont exploitables pour le calcul de l'indice cubital par nos algorithmes. On distingue 3 origines différentes pour les erreurs : la détection échoue car une intersection n'est pas détectée et n'apparaît pas dans notre liste de points potentiels, la séparation en sous-groupes pour calculer la moyenne n'est pas satisfaisante, ou bien l'indice cubital calculé est aberrant.

Parmi les 50% d'ailes exploitables, environ 22% ne passent pas l'étape suivante ; le calcul informatique et le calcul manuel donnent des conclusions différentes. (ici pour 2 individus) 33% donnent la même conclusion sur l'indice cubital, et les autres donnent un résultat non concluant, car lorsque l'indice cubital est compris entre 1,55 et 2, il n'est pas possible de conclure sur l'espèce de l'abeille grâce à ce critère.

Ces calculs sont faits sur une seule ruche de 18 individus donc sur un petit échantillon, ce qui réduit leur fiabilité, mais on peut déjà conclure que notre algorithme n'est pas assez robuste pour traiter toutes les ailes efficacement et pour déterminer le taux d'hybridation d'une ruche.

Interface graphique

A. Présentation générale

Afin de satisfaire les attentes des apiculteurs, nous proposons de développer une interface graphique pour faciliter l'utilisation du programme par les usagers qui ne maîtrisent peut-être pas les langages informatiques. Nous allons donc utiliser le même langage que dans l'ensemble du projet, c'est-à-dire Python, et nous nous aiderons de la librairie Tkinter [tkinter] pour la construire. Le langage Python nous permet d'offrir une possibilité gratuite aux apiculteurs pour traiter leurs données.

B. Présentation de l'algorithme

L'application est pensée de la manière suivante. L'apiculteur fournit une image quadrillée comme présentée sur la figure n°1.1. Pendant le temps de traitement des données (découpage de l'image en plus petites images, filtrage, repérage de tokens ciblés, analyse des points d'intérêt et calcul des éléments finaux), un écran d'attente sera disponible mais il ne permet pas de savoir à quel pourcentage du traitement nous nous plaçons. Ainsi, même si le traitement est un peu long, c'est-à-dire une centaine de secondes, il ne faut pas s'en inquiéter, si un problème apparaît, il sera signalé.

L'idée est qu'une page de résultats apparaîtra par la suite fournissant les données suivantes à partir des traitements faits et des formules de calcul fournies par les apiculteurs dans le fichier Excel cité précédemment :

- La toile d'araignée représentant l'indice cubital en fonction de l'indice discoïdal
- L'histogramme de l'indice cubital
- L'histogramme des classes Dreher avec une courbe de Gausse
- L'histogramme de Hantel entre 0.60 et 1.20

Les résultats auront l'aspect comme représenté sur la figure n°4.1, ceci étant une des exigences de nos clients. Un code a déjà été développé afin de créer des images qui représentent les attentes précédentes. Un autre code permet l'affichage de ses résultats dans l'interface de l'application.

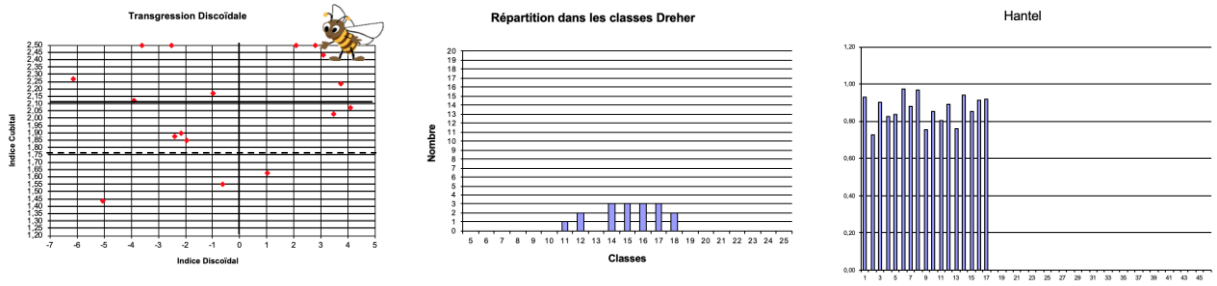


Figure n°4.1 : représentation des résultats attendus

C. Utilisation du code

Lors du lancement de l'application, l'utilisateur voit l'écran de la *figure* n°4.2.a. L'interface homme-machine sera presque uniquement faite par l'intermédiaire de boutons. Ainsi, en appuyant sur "Commencer le traitement", l'écran 2 avec les instructions et une représentation des résultats finaux (qui nécessitera une mise à jour lorsque nous aurons la possibilité de les fournir) apparaît. En appuyant sur "Commencer pour de bon cette fois-ci", une page permettant de choisir son fichier image (figure n°4.2.c) dans son ordinateur apparaît. Cette page lui permet de naviguer facilement au sein de l'ordinateur, comme l'utilisateur le ferait en parcourant simplement ses fichiers.

Une fois le fichier choisi, le traitement commence. Il peut y avoir des erreurs lors du traitement.

Dans ce cas, le traitement s'arrête et indique à l'utilisateur que le traitement de cette image choisie est impossible (figure n°4.2.e). Puis, se trouvant sur l'écran 2 (figure n°4.2.c), l'usager se trouve dans la possibilité de relancer un traitement comme expliqué précédemment. S'il n'y a pas d'erreur, et que le traitement se réalise correctement, alors à la fin du traitement, l'utilisateur sera averti et il pourra alors consulter les résultats dans le dossier créé pour cela (figure n°4.2.d).

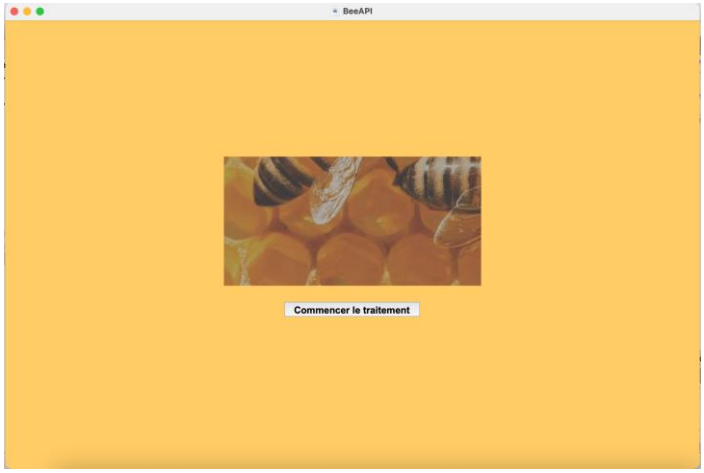


Figure n°4.2.a : écran d'accueil de l'application

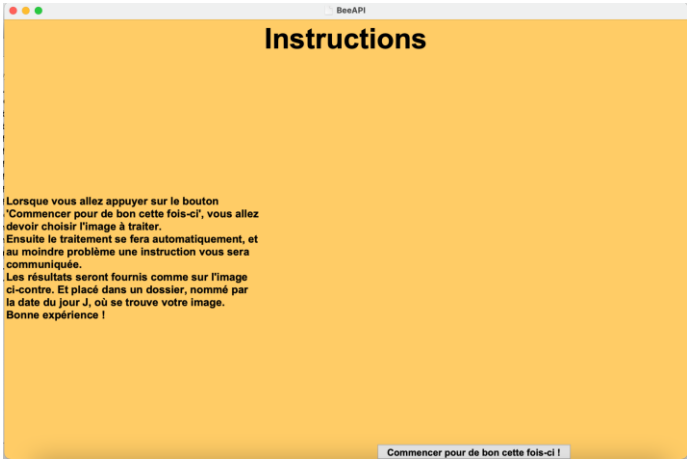


Figure n°4.2.b : écran 2 de l'application, instructions

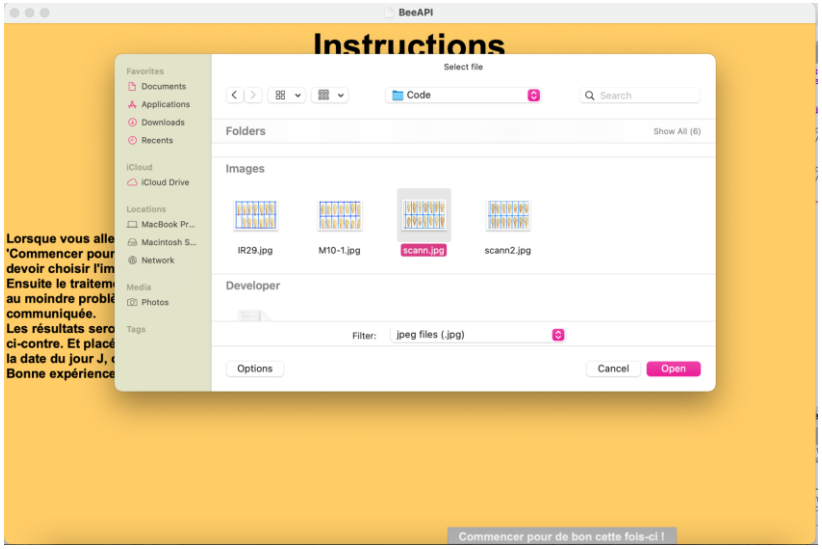


Figure n°4.2.c : écran 3 de l'application, choix de l'image à analyser par parcours des dossiers

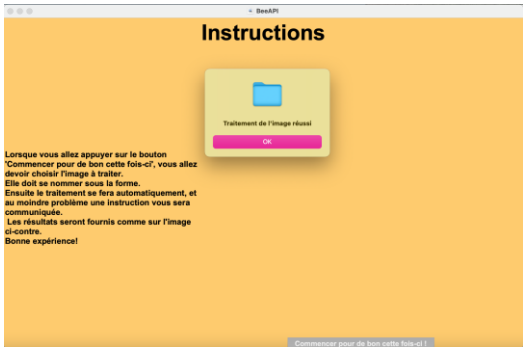


Figure n°4.2.d : affichage lors d'une réussite de traitement

Figure n°4.2.e : affichage lors d'un échec de traitement

Figure n°4.2 : écrans de l'application

CONCLUSION

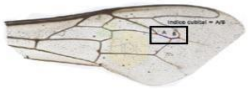
Le projet qui a été mené cette année, bien qu'inachevé, nous a été particulièrement enrichissant puisqu'il nous a permis de nous familiariser avec divers outils de traitement d'images tels que la segmentation, le filtrage, la reconnaissance de pattern et l'extraction de l'information, le tout, pour répondre à une problématique concrète et passionnante.

Bien que l'ensemble des algorithmes soient optimisables, l'ensemble des grands axes du traitement ont été programmés et les résultats (développés dans les parties précédentes) sont encourageants pour l'année prochaine.

Le poster suivant permet de résumer les grandes étapes du traitement réalisé et l'état actuel de l'algorithme.

ANALYSE BIOMÉTRIQUE DES AILES D'ABEILLES

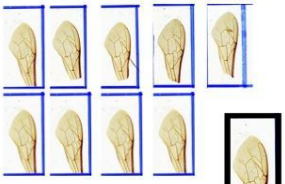
Obtention de l'indice cubital



Etape 1

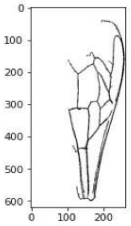
Segmentation
Isolation des ailes

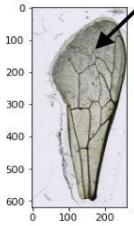
Image originale



Etape 2

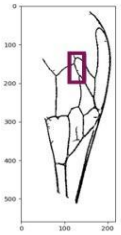
Filtrage

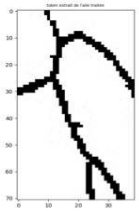




Etape 3

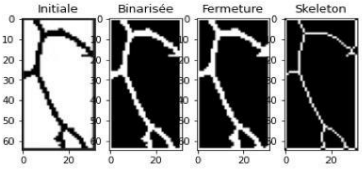
Extraction des motifs





Etape 4

Traitement des motifs extraits



Indice cubital

Barbier Arthur - Bauchat Alice - Poulic Chloé - Segal Léontine

BIBLIOGRAPHIE

[1] Rapport de projet réalisé par les élèves de la filière ASI de l'ENSE3
[2] **Eysteinn Mar Sigurðsson, Silvia Valero, Jon Atli Benediktsson, Jocelyn Chanussot, Hugues Talbotd, Einar Stefansson.** *Automatic retinal vessel extraction based on directional mathematical morphology and fuzzy classification*, 1 Octobre 2013 [janvier]
[3] **Friedrich Ruttner, Eric Milner, John Dews.** *Abeille noire Apis mellifica mellifica Linnaeus* [janvier]
[4] Biométrie 3 : déterminer la race d'abeilles par analyse des ailes, disponible sur : *https://www.youtube.com/watch?v=pPMVvKCjWX0* [janvier]
[5] Sujet de BE Phelma : *Reconnaissance de personnes par analyse automatique des empreintes digitales* [mars]
[6] **Hasnaoui Nassim Aboubakr.** *La reconnaissance automatique des empreintes digitales* [mars]
[7] Exemple d'utilisation de la fonction watershed *https://scikit-*

image.org/docs/dev/auto_examples/segmentation/plot_expand_labels.html#sphx-glr-auto-examples-segmentation-plot-expand-labels-py
[8] Fichier explicatif de la fonction K.Means *https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html*
[9] Documentation de la fonction de segmentation par la méthode watershed *https://scikit-image.org/docs/dev/api/skimage.segmentation.html#skimage.segmentation.watershed*
[10] Documentation des fonctions de mesures d'objets *https://scikit-image.org/docs/dev/api/skimage.measure.html*
[tkinter] Documentation Tkinter pour réaliser des interface graphiques en langage Python *https://docs.python.org/fr/3/library/tkinter.html*
[11] https://scikit-image.org/docs/dev/auto_examples/registration/plot_register_rotation.html

Annexes

Annexe : Fonctionnement de l'algorithme Watershed

(Extrait TP - SICOM 2A - Lidar, Jocelyn Chanussot)

Principe

Le concept de l'algorithme bassin versant (en anglais : watershed) ou de ligne de partage des eaux est assez simple. Il revient à considérer une image comme un relief en trois dimensions, dont l'altitude serait donnée par la valeur de chaque pixel.

Les différentes "cuvettes" topographiques que contient l'image peuvent être identifiées en simulant une inondation progressive. On fait monter progressivement la hauteur de l'eau sur l'ensemble de l'image. Lors de la montée de l'eau du niveau h à h + 1, considérons un pixel i qui se retrouve "noyé" car son altitude (sa valeur) est comprise entre h et h + 1. Ce pixel peut se trouver dans trois cas de figure :

— il n'est adjacent à aucun pixel déjà noyé : c'est donc le point minimum d'une nouvelle cuvette

— il est adjacent à un ou plusieurs pixels appartenant à une même cuvette : il en fait donc partie

— il est adjacent à des pixels appartenant à différentes cuvettes : il fait donc partie d'une ligne de partage des eaux (contour de ces cuvettes). On y construit alors une digue virtuelle pour empêcher la fusion des cuvettes qui viennent de s'y rencontrer.

En noyant progressivement l'ensemble de l'image, on aboutit ainsi à une segmentation de celle-ci en un nombre fini de cuvettes auxquelles s'ajoute un ensemble de pixels constituant les lignes de partage des eaux (figure n°).

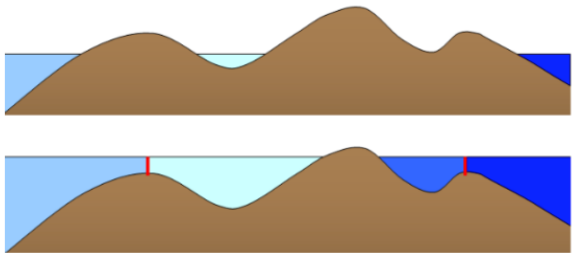


FIGURE 3 – Algorithme bassin versant : deux étapes de l'inondation simulée d'un relief en coupe (en marron). Sur l'image du haut, trois bassins versants (tons de bleu) sont en cours de remplissage. Sur l'image du bas, la montée des eaux continue de remplir les bassins et deux points de partage des eaux ont été identifiés lors de la rencontre entre deux cuvettes adjacentes. Ils sont matérialisés sous forme de digues rouges.

Afin d'obtenir une bonne segmentation (identification d'éléments distincts au sein d'une image) par cet algorithme, il peut être nécessaire d'appliquer auparavant une transformation à l'image d'origine (gradient, laplacien...) pour que les contours des objets apparaissent bien comme des lignes de partage des eaux.